コンピュータの原理(2)

～マシン語とは何か ～

Ver. 1.0

リナックスアカデミー矢越昭仁

2014 年 11 月 15 日

プログラムを用意することで、いろんな事をこなしてくれるコンピュータですが、その指示はコンピュータの言葉であるマシン語によって行われます。このマシン語とはどんなものか、コンピュータ（CPU）の歴史や仕組みを交えて解説します。
これでカタログに載っている GHz, 32bit/64bit AMD/Intel/ARM などの謎が解けるはずです。

目次

はじめに

いつのまにかインターネットが普及し、コンピュータエンジニアや科学者以外でも手軽に利用できるようになりました。最近では社会インフラとしてのインターネットが、存在感を増しています。

このコースでは、今では生活必需品となった「インターネット」が公開される前夜から、現在までを振り返ります。そしてホンの少しだけ未来を予想し、仕事に役立つヒントを提供します。

表記について

この資料では以下の表記としています。

・フォント

コンピュータの操作および設定ファイルはクーリエフォント(タイプライター風)を用います。

```
search t123006.la.net
nameserver 10.20.123.6
```

・プロンプト

コマンド入力例がある場合は、先頭はプロンプト($または#)で始めます。

$ は一般ユーザでの操作、#はルートユーザでの操作を表します。なおユーザ切り替え(su)の表記は省略しています。

・強調(ボールド)

コマンド入力では、キーボードから入力する場合を、設定ファイルの場合は修正箇所など特に強調したい場合に**ボールド**を使います。

```
$ date
Mon Mar 5 12:32:41 JST 2012
```

```
DEVICE=eth0
NM_CONTROLLED=yes
ONBOOT=yes
```

オンラインバックアップ

矢越が実施した IT 特別講座の資料(補足資料、例題等含む)は、以下の URL にて掲載しています。この URL はリナックスアカデミー会員限定となっていますので、それ以外への再配布・再掲載は遠慮ください。

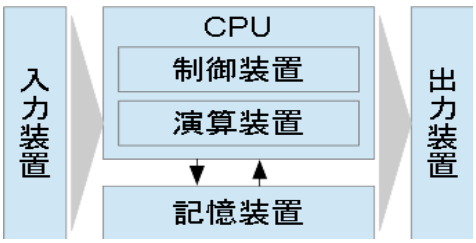
http://ycos.sakura.ne.jp/LA

また講座・資料への質問、要望は下記までメールをお願いします。

ycos001@yahoo.co.jp

CPU とは

コンピュータの構成要素



現在主流となっているコンピュータの構成要素は、入力装置、出力装置、演算装置、制御装置、記憶装置の5つからなり、コンピュータの5大装置と呼ばれます。このうち演算装置と制御装置をまとめて、中央処理装置 CPU: Central Processing Unit といいます。古くは沢山の電子部品を組み合わせた装置でしたが、最近では LSI という単一部品にまとめられています。そのため、昔は CPU は大型の装置、単一の LSI は MPU: Micro Processing Unit と使い分けていました。

各装置の概要は以下の通りです。

- 入力装置
  外部からコンピュータへデータや指示を行うための装置。キーボード、マウスなど。最近ではスマートフォンの普及で GPS、カメラ、マイクなども入力装置として普及している。
- 出力装置
  処理結果を外部へ伝える装置。ディスプレイ、プリンタ、スピーカなどを指す。
- 制御装置
  他の装置に対し、処理のタイミングや順番を指示する装置。殆どの CPU ではクロック周波数にもとづき処理の速度が決まる。一般的な CPU の処理は、記憶装置から命令を取り込むフェッチ、命令を解析するデコード、実際に処理を行うエクゼキューション（実行）の3つのステップがある。
- 演算装置
  実際に計算を行う装置。数値演算だけでなく、AND、OR、XOR といった論理演算も行うため ALU: Arithmetic Logical Unit とも呼ばれる。また三角関数などの科学技術演算を専用に行う FPU: Floating Point Unit を別部品とした構成もある。
- 記憶装置
  プログラムやデータを保持する装置。CPU と直接接続されたものを主記憶装置とよび、一般的なコンピュータではシステムが停止するとその内容が失われる（揮発性）。そのため、磁気ハードディスク装置などの二次記憶装置を伴う。

この5つの分け方は日本の情報処理学会でよく用いられる方法ですが、国際的には演算装置と制御装置だけに着目し、数学的なモデルを研究するオートマトン automaton とう分野もあります。1936 年のアラン・チューリング[1]（英数学者 Alan Mathison Turing, 1912-1954）の万能計算機械に端を発するものです。

プログラム内蔵方式

先の基本構造は一般的に作者ジョン・フォン・ノイマン(ハンガリー・米数学者 John von Neumann 1903-1957）の名を取ってノイマン型と呼ばれ、今日でも主流となっています。このアイデアを最初に実現したのは 1951 年に部分稼動した EDVAC（Electronic Discrete Variable Automatic Calculator、電子的な離散変数自動計算機、当時はアナログ計算機も使われていたためか）とも言われています。実際には

[1].現在コンピュータ理論の祖といわれ、アルゴリズムを明確にしチューリング・マシンを提唱。計算機科学分野で世界最高の権威である ACM チューリング賞に名を残す。

ENIAC のメンバーのチームに、後からノイマンが参画し特許を取得したため先に同様のアイデアで作られた Manchester Mark I(1949)よりも有名になりました。

プログラム内蔵方式(Stored-Program concept)はプログラムを主記憶に配置し、実行中にプログラム自身を書き換える事ができるもので、汎用的なプログラムをコンパクトに作る事ができるという特徴があります。反面、以下の欠点が指摘されています。

- ノイマンの隘路(Von Neumann bottleneck)
  プログラムの読取、データの読み書きがすべて主記憶を経由するため、CPU とメモリの間の転送速度によりシステム全体のパフォーマンスに影響する。
- プログラムの改ざん(Data Execution)
  確保されたデータ領域に必要以上のデータを書き込む事でプログラム領域を破壊してしまうバッファオーバーランや、データ領域にプログラムを仕込み不正な処理を行うデータ実行など脆弱性の温床となっいる。

ノイマンの隘路を改善するために、CPU のとメモリの間に小容量高速なメモリを搭載する方法があります。キャッシュメモリ(cache memory, 隠しメモリ)と呼ばれ、CPU 内に CPU と同じ速度で動作できる高速なメモリを搭載する事が一般化しています。速度の違うキャッシュを多段階に用意するものも増えています。CPU に近いほうから L1 Cache、L2 Cache というようにレベルで呼称し、近年では L3 までも CPU に内蔵するものがあります。

もうひとつの欠点、特にインターネットが普及した今日ではプログラム改ざんといった脆弱性に対し、OS やハードウエアのレベルでデータ領域を実行しない NX(Not Execute flag)、DEP(Data Execution Prevention)といった処置がとられています。

ワード(語)とアドレス

プログラムもデータも同じ主記憶装置に数値として格納されていますが、それを取り出し処理する単位を CPU の語長(ワード長)と呼びます。ALU で一度に計算できる桁数でもあります。1970 年にインテルが発表した世界初の MPU　i4004 はワード長 4bit でしたが、70 年代中ごろには i8080、モトローラの MC6800 など 8bit となり、80 年代には 16bit、90 年では 32bit、2010 年には 64bit が主流になりつつあります。1 Byte が 8bit として定着した 1980 年以降は 8 の倍数がほとんどですが、それ以前は 8 進数で使い勝手の良い 6 の倍数である 12bit, 18bit, 24bit, 28bit なども存在しました。

8bit 時代には、実際に計算を行う部分であるアキュムレータ(Accumulator、累算器)が 8bit でも、メモリの座標を示すアドレッシングは 16bit というように、計算できる桁と、メモリーの位置を示す桁が異なる物もありましたが、32bit 以降ではほぼ同じ CPU が主流になっています。8bit 時代に主流だった、Z-80, MC6809 などはアキュムレータは 8 ビットなので計算できる数値の範囲は 0～255 または-128～127 となります(2^8=256 通り)。

念のため 2 進数の並び(ビット列、ビットパターン)と 10 進数の割り当て方法をおさらいすると、符号なしと、1 の補数、2 の補数の 3 種類があります。1 の補数は人からみれば分かりやすいですが、＋0と、－0という矛盾が生まれますし、電気回路で処理する場合は 2 の補数の法が簡単に実装できるのでした。

- 1の補数
  最上位(もっとも左側上位)のビット MSB: Most Significant Bit を符号に見立て、0 のときは正、1 の時は負というように表現する方法。
- 2 の補数
  MSB よりひとつ多い桁(たとえば 8bit であれば 100000000(2)=256)から、整数を引いた値を負

の数とする方法。回路の場合はビットパターンを反転させ、1 を加算する方法が用いられる。

ビットパターン	符号なし	1の補数	2の補数
'0000.0000'	0	0	0
'0000.0001'	1	1	1
'0000.0010'	2	2	2
'0111.1111'	127	127	127
'1000.0000'	128	0	-128
'1000.0001'	129	-1	-127
'1000.0010'	130	-2	-126
'1111.1110'	254	-126	-2
'1111.1111'	255	-127	-1

アドレスはこれらの数値が格納される主記憶の場所を示す値で、連続した符号なし整数で表します。先の 8bit CPU では、256 バイトしかプログラムやデータが扱えないのでは不便だったので、アドレス部分は 16bit = 2^{16}=65,536 個が扱えるよう変則的な設計でした。

このアドレスで示される主記憶に1 Byte 単位で数値を格納するものをバイトマシーン、ワード長で格納するものをワードマシーンといいます。ワードマシーンは主記憶からデータを取り出したり格納したりする機構が複雑になるため、現在はバイトマシーンが主流を占めます。

例）16bit のバイトマシンと、ワードマシンでアドレスに格納されるデータのイメージ（16 進表記）

	ワードマシン	バイトマシン
アドレス	データ	データ
0001	1234	12
0002	5678	34
0003	9ABC	56
0004	DEF0	78

バイトマシンでは、CPU のワードをバイトに分割して格納するため、複数のアドレスにワードがまたがる事とになります。このときワードの 上位と下位をどのようにアドレスに割り振るかが問題になります。これをバイトオーダとよび、データの 上位から順に格納する方をビックエンディアン[2]、データの下位から格納する方をリトルエンディアンと呼んでいます。

例えば1234というデータを、12、34と格納するのはビッグエンディアン、34、12と格納するのはリトルエンディアンとなります。

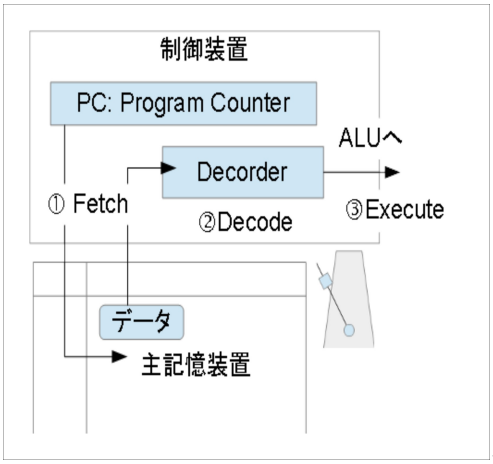
TCP/IP ではネットワーク上のバイトオーダはビックエンディアンに統一されて、ネットバイトオーダと呼びます。CPU それぞれのオーダはホストバイトオーダとよび、データを送る際にはホスト・ネットバイトオーダ変換を行う必要があります。具体的には関数、htonl（unsigned long=32bit データのホスト→ネットワーク変換）、htons(unsigned short=16bit データのホスト→ネットワーク変換)、ntohl（同様に 32bit ネットワーク→ホスト変換）、ntohs（同様に、16bit ネットワーク→ホスト変換）が用いられます。

[2] Big Enddian, Little Endian は「ガリバー旅行記」で登場する、ゆで卵を丸い方から食べる部族と、尖った方から食べる部族にちなんでいます。

Java は CPU に関係なくビックエンディアンで動作するので、このような違いを意識する必要はありません。また最近の CPU では、設定によりバイトオーダーを切り替えられる物もあります。

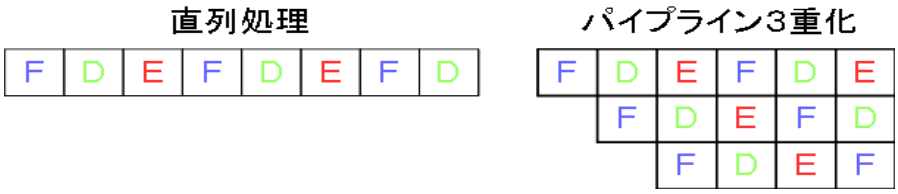
制御装置の役割

制御装置はメモリーからデータを取り出し、内容を解析し他の装置に処理指示する事と、それを含め処理のタイミングを統制する2つの機能からなります。解析は以下の 3 つの手順で行われます。



1. Fetch
   プログラムカウンタにより示されるメモリのアドレスを元にデータを取り出し、デコーダへ送ります。
2. Decode
   データの内容を解析し、命令か引数か。加減乗除の数値計算か、AND、OR、XOR といった論理演算か等を判断します。また PC を更新します。
3. Execute
   解析した内容を元に、ALU 等へ作業指示を行います。

この動作は、一定の周期により順に処理されてゆきます。このリズムをクロック周波数といい CPU の仕様書に Hz で表されています。たとえば1 MH zであれば、1 秒間に 100 万回1～3の処理が行われます。
実際の CPU では、この1～3の流れをパイプラインとよび、多重化することで処理を効率化しています。



例えばパイプラインを3つ用意すると単純計算で、ほぼ毎サイクル実行できることになり 1MHz で100万回の処理が行える事になります。1秒間に実行できる処理（命令）を MIPS: Million Instructions Per Second といい、処理能力の指標になっています。

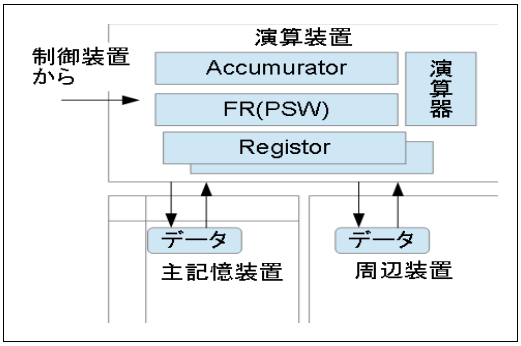
命令はオペコード(Opecode、Operation code)とよばれ、続く引数はオペランド（Operand)と呼びます。命令によっては引数を持たないものや、複数の引数を持つものまで様々です。これらが一般にマシン語と呼ばれるもので、CPU 型式ごとに異なります。CPU で扱う事の出来る語彙すべてを命令セット(Instruction Set)と呼びます。

PC は基本的にはワード長が加算されますが、命令の種類やオペランド数により加算幅が異なります。これらが異なるとパイプラインの再計算を行ため、ワード長を同一にするなど、処理を簡略化し、クロック周波数を向上させる技術を RISC：Reduced Instruction Set Computer と呼びます。1990 年台に非常に流行しましたが、半導体技術が向上し回路を簡略化しなくても周波数向上が可能となったため当時ほどの勢いは感じられません。また従来からの命令セットを持つ CPU は CISC：Complex Instruction Set Computer と呼ぶようになりました。

今日のプロセッサ(Debian が対応しているもの)

CISC	RISC	その他
amd64(x86_64), i386/686, System Z(ESA/390,)	ARM, PowerPC, SPARC, MIPS	IA-64(Itanium)

処理装置の役割



処理装置の各機器は制御装置の指示に従い、演算を行います。演算に必要なデータとその結果はアキュムレータ(Accumurator、累計器)に貯えられます。演算の処理についての状況は FR(Flag Registor)または、PSW(Program Status Word)に収められます。この状況には、計算結果が正・負、桁あふれエラー、ゼロ割エラーなどを示す値が入ります。多くの場合は各ビットに意味を持たせ、その事象が発生した場合にフラグを立てます(当該位置の二進数を0→1に変更する)。

主記憶装置や周辺装置からデータを読書きする場合も、一旦レジスター(Registor、置数器)を経由して処理されます。汎用的に利用できるレジスタは GR: General Registor と呼びますが、いくつかは特別な用途に用いられることがあります。以下の例は情報処理試験問題の COMET II です。

略号	名称、意味
GR	General Register 0～7 汎用的に利用可能なレジスタ。計算結果やメモリ、周辺装置とのやり取りを行う
PR	Program Register 又は PC: Program Counter とも。 実行する命令があるメモリのアドレスを保持
SP	Stack Pointer スタック構造のデータの最上段にあたるアドレスを保持

[3]「試験で使用する情報技術に関する用語・プログラム言語など」Ver2.2(2012 年 5 月版)

周辺機器へのアクセス

すでに説明したとおり、演算装置のうち直接主記憶とアクセスできるのはレジスタになります。主記憶からレジスタにデータをコピーすることを Load、逆にレジスタのデータを主記憶へ書き込むことを Store と呼びます。このままでは周辺機器とやりとりができません、そこで周辺機器とやりとりする方法が必要となり、2つの方法があります。

1. メモリマップド I/O (Memory mapped Input / Output)
   周辺装置のデータ領域を特定のメモリ領域に割り当てて処理をする方法。特定の領域を周辺機器に割り当てた後は、メモリへのアクセス同様 Load、Store 命令だけで処理が可能。ただしマシン語を一瞥しただけでは、周辺機器への操作を行っているのか主記憶とのやり取りかは不明となる。

[3] URL は http://www.jitec.ipa.go.jp/1_13download/shiken_yougo_ver2_2.pdf

2. ポートマップド I/O (Port mapped I/O)
   周辺機器へはメモリと異なるポートと呼ばれる領域を割当て、CPU から周辺機器へは OUT、周辺機器から CPU へは IN 命令を使う。周辺機器と主記憶でプログラム方法が異なり煩雑になるが、主記憶を他の用途に利用されることがない。Intel 系の CPU でよく用いられる。

実際のプログラミングでは、OS がその差を吸収する仕組みを持っているため、上記のような方式の違いを意識する事は殆どありません。

マシン語の実際

CPU 動作確認

では実際に CPU がどのようにメモリーからデータや命令を取り出し、実行しているかを確認してみましょう。今回は、情報処理試験用に用意された COMET II を例としました。COMET II は実在しませんが、試験用に詳しくマシン語が解説されていて、しかも比較的単純なので勉強にはちょうどよいでしょう。

情報処理試験の資料から関連する部分を以下に抜粋しました。

1. **命令語の構成**
   命令語の構成は定義しないが，次のような構成を想定する。ここで，OP の数値は 16 進表示で示す。

15 11 7 3 0 15 ← 0 ビット番号

第1語				第2語	命令語長	命令語とアセンブラとの対応		
OP		r／r1	x／r2	adr		機械語命令		意味
主OP	副OP							
0	0	—	—	—	1	NOP		no operation
1	0				2	LD	r,adr,x	load
	1				2	ST	r,adr,x	store
	2				2	LAD	r,adr,x	load address
	4			—	1	LD	r1,r2	load
2	0				2	ADDA	r,adr,x	add arithmetic
	1				2	SUBA	r,adr,x	subtract arithmetic

では、実際のマシン語（16 進数データの並び）を見て、CPU の動作を確認しましょう。

問）COMMET II は16ビット CPU（ビックインディアン）で、以下の32バイトのデータがあり、アドレスの先頭から実行する場合、どのような処理を行うか検証せよ。但し、システム停止命令は FFFF（16）とする。

Addr	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
0000	10	00	00	10	20	00	00	12	11	00	00	14	FF	FF	00	00
0010	00	01	00	02	00	00	00	00	00	00	00	00	00	00	00	00

この表は、アドレス 0000 番地から、001F 番地までが表示されています。例えば 0000 番地のデータは 10、0008 番地のデータは 11 とう事を表しています。スタートは先頭なので 0000 番地から Fetch、Decode をやってみましょう。

まず、0000 番地のデータは 10 なので、命令語の構成をみると、主 OP（最初の1バイト）が1で始まり、副 OP（次の1バイト）が 0 ということで、Load 命令だという事が分かります。またこの時点で、命令語長が 2 ワードつまり、16bit × 2 = 4 バイトだという事が判明します。よって、最初のオペコードは「10 00」、オペランドは「00 10」となります。
再び、命令表で確認すると、第 1 語の後半部分は「0」ですので、レジスタを表す r は 0、変位を表す x も 0 という事が分かります。さらに第 2 語は addr であり、その値は「0010」という事が分かります。
よって、最初の命令は「レジスタ 0 へ、アドレス 0010 の内容を取り込め（Load）」となります。さらにアドレス 0010 には、「0001」とあるので、結果として「GR0←1」が実行された事になります。

最初の命令が2ワード（4 バイト）だったので、つぎはアドレス 0004 から Fech を開始します。「20 00 00 12」は、GR0 に、アドレス「0012」にあるデータを加算せよという事になり、結果としては「GR0←GR0+2」となります。更に続く「11 00 00 14」は、「GR0 の内容を、アドレス 0014 へ書き込め(Store)」とう指令、最後は「FF FF」で、処理完了となります。

もしGR0ではなく、GR1を使うのであれば、各命令は「10 10 00 10, 20 10 00 12, 11 10 0014, FF FF」となります。この様に数字の羅列がマシン語といわれ、CPUが直接理解(Decode)できるものになります。

アセンブラ言語

マシン語はCPUが直接理解できる唯一の言語ですが、あまりにも人にとって分かりずらいものです。また、データを格納する場所を「0010」といった数値で指定されると、プログラムが大きくなった場合や、前提が異なり全く別の領域へ移動した際に修正が難しいものとなります。そこで、マシン語の命令に1対1で対応した記号(ニーモニック、Mnemonic、表意記号)や、アドレスに名前を付け(ラベル、Label)人に分かりやすく変換したプログラミング言語が登場しました。先のマシン語を、COMETのアセンブラ言語であるCASL II[4]で書き直すと、次のようになります。

```
CASLⅡ Assembler -V2.00(2005/05/27)- ( Date : 2014/9/16 )
                        1: ; Sample program
                        2:     START
  0000:1000-0007R       3:     LD      GR0,DATA01
  0002:2000-0008R       4:     ADDA    GR0,DATA02
  0004:1100-0009R       5:     ST      GR0,DATA03
  0006:8100             6:     RET
  0007:0001             7: DATA01              DC      #0001
  0008:0002             8: DATA02              DC      #0002
                        9: DATA03              DS      1
                       10:     END

  Label:DATA01    Def(Loc):    7(0007) Ref:    3
  Label:DATA02    Def(Loc):    8(0008) Ref:    4
  Label:DATA03    Def(Loc):    9(0009) Ref:    5
オブジェクトプログラムを la01.obj へ出力しました。
END CASLⅡ Assembler.
```

ここで使用しているラベルDATA01～03が、自動的にプログラムの最後のアドレスから採番されていることが分かります。

アドレッシングモード

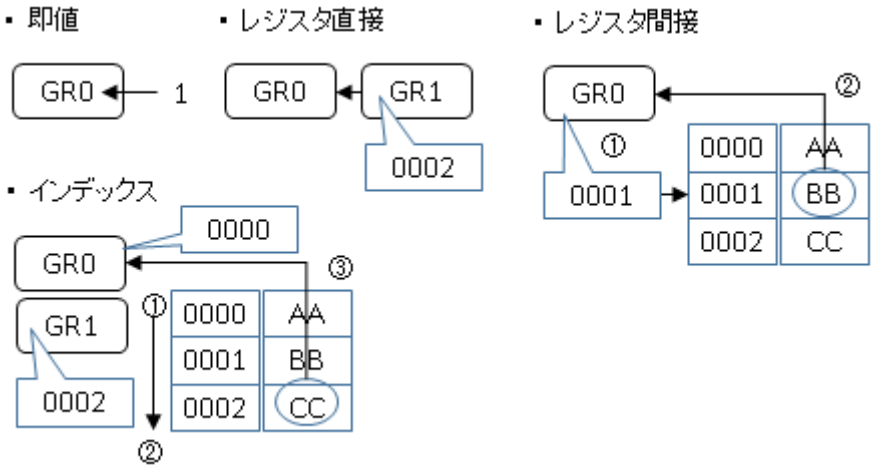
先の例でもわかるように、マシン語において命令に対象となるレジスタまで含まれているため、引数に当たるオペランドは主にアドレスをが用いられます。例えば、「LD GR0,XX」、「LD GR1,XX」はオペコードがそれぞれ10、11とマシン語としては別物にが、続くオペランドはXXで同じになります。

そこでオペランドのXXが何を意味しているかによって動作がかわります。この事をアドレッシングモードと言います。主なアドレッシングモードには次の物があります。

- 即値アドレッシング(Imediate Addressing)
  続く数値をそのまま採用する。例えば0001は、数値0001として扱う。

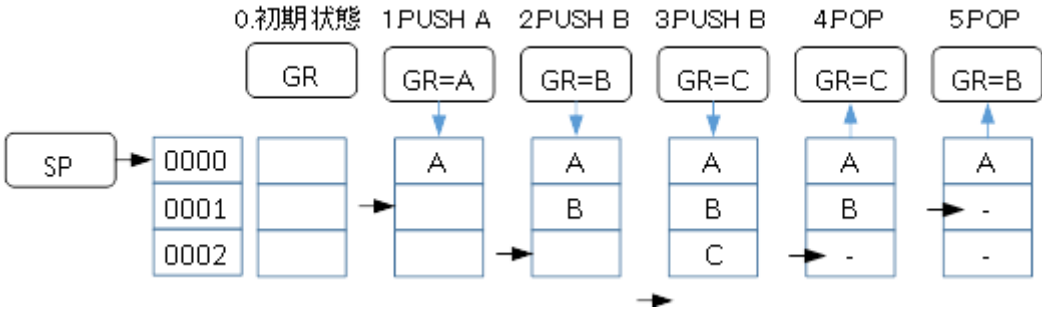
[4] シミュレータはIPAよりダウンロード可(HOME>情報処理技術者試験>CASL IIシミュレータ…)
https://www.jitec.ipa.go.jp/1_20casl2/casl2dl_001.html

- 直接アドレッシング（Direct Addressing）
  続く値で示されたアドレスに記載されたデータを値とする。先のサンプルの例。
- レジスタ直接（Register direct addressing）
  続く値で示されたレジスタに記載された内容を値とする。例えば 1 であれば、GR1 の値を採用する。
- レジスタ間接（Register indirect addressing）
  続く値で示されたレジスタに記載されたアドレスを値とする。例えば 1 であれば、GR1 を参照し、そこに記載されたアドレスに格納されたデータを値とする。
- インデックス参照（Index adressing）
  既定のアドレスに対し、指定した変位を加えた先にあるアドレスのデータを値とする。例えば規定が GR0（=0000）で、指定された値が GR1(+2)であれば、アドレス 0002 に格納されたデータを値とする。



プログラム言語では、このようなアドレッシングの事をポインタと呼び複雑なデータ構造を実現しています。特に C 言語で多くの人がつまづく箇所だと言われています。

特にインデックス参照はスタックと呼ばれるデータ構造でよく利用されます。既定となるアドレスを SP:Stack Pointer に用意しておき、レジスタの値を重ねていく方法です。PUSH で積み上げ、POP で取り出します。例えば、SP:0000 の時に A、B、C を順に PUSH し、次に POP するといった操作は以下の様になります。



最後に

今回ご紹介したように、マシン語は CPU（ハードウエア）が直接理解できる唯一の言語です。それをハードウエアに依存せずに、簡単にプログラム出来るよう開発されたのがプログラム言語です。アセンブラもプログラム言語ですが、よりハードウエア（原始）に近いところから低級言語と呼ばれています。

今回の講座を通じ、CPU の動作原理やプログラムの本質（必要最小限の機能である命令＝オペコード）を理解することで、セキュリティの脆弱性といった最新の問題が身近に感じて頂ければと思います。

アセンブラに興味を持たれた方は、IPA の情報処理試験用のツールが役立ちます。特定のメーカに依存しないよう、また初心者に理解しやすいよう制定されたハードウエア COMET II と、そのアセンブラである CALS II は、IPA ホームページで仕様が公開されています。
また Java で作成された CALS シミュレータは、アセンブラのソースを COMET II のマシン語に変換してくれます。また実行させることも出来るので是非試してみてください。